2020 자율주행 교육

WeGo 위고 주식회사

목차

WeGo

01

Classification and Regression

WeGo

01. Classification and Regression

- 출력층 설계

→ 신경망은 분류(classification)와 회귀(regression)에 모두 사용이 가능

→ 분류는 데이터가 어떤 클래스에 속하는지 찾는 문제

→ 회귀는 입력 데이터에 대해 수치를 예측하는 문제(Linear regression)

→ 사용 방식에 따라 출력층의 활성화 함수를 다르게 사용해야함

→ 회귀에서는 결과를 그대로 사용하는 항등 함수(출력 그대로 사용)를 적용

→ 분류의 경우 Softmax 함수를 사용하여, 분류에 적용

- Softmax function
- 아래와 같은 수식을 통해, 각 분류별 점수(확률)을 계산
- 이 후, 높은 값을 가지는 항목으로 분류 진행
- Softmax의 분자는 지수함수 형태이므로, 프로그래밍 시 오버플로우 위험이 있음
- 따라서 아래와 같은 변형을 통해 사용
- 사용하는 C'의 경우 입력으로 들어가는 a_i중 최대값을 이용하는게 일반적

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)} = \frac{C exp(a_k)}{C \sum_{i=1}^{n} \exp(a_i)}$$

$$= \frac{\exp(a_k + \log C)}{\sum_{i=1}^{n} \exp(a_i + \log C)} = \frac{\exp(a_k + C')}{\sum_{i=1}^{n} \exp(a_i + C')}$$

- Softmax function

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)} = \frac{Cexp(a_k)}{C\sum_{i=1}^{n} \exp(a_i)}$$

$$= \frac{\exp(a_k + \log C)}{\sum_{i=1}^{n} \exp(a_i + \log C)} = \frac{\exp(a_k + C')}{\sum_{i=1}^{n} \exp(a_i + C')}$$

```
def softmax(a):
    c = np.max(a)
    exp_a = np.exp(a-c)
    sum_exp_a = np.sum(exp_a)
    y = exp_a / sum_exp_a

    return y
```

02

MNIST Data

02. MNIST Data

- MNIST Data
- 손으로 쓴 숫자들로 이루어진 데이터
- 딥러닝 기초를 위해 자주 사용되는 데이터
- git clone https://github.com/k9632441/MNIST_Dataset.git
- 명령어를 통해 Data를 받을 수 있음
- Dataset 아래의 MNIST.py 파일의 load_mnist를 통해 데이터를 받아서 확인 가능

02. MNIST Data

- load_mnist는 입력 인수로 normalize, flatten, one_hot_label을 받음
- Normalize의 경우 입력 이미지의 픽셀 값을 0~1 사이로 정규화
- Flatten은 1 x 28 x 28의 이미지를 1 x 784로 변환하여 1차원화
- One_hot_label의 경우 False일 경우, 정답에 해당하는 값(7 또는 2) 등이 바로 나오며, True일 경우, [0, 0, 0, 0, 0, 0, 0, 1, 0, 0]와 같은 형태로 배열로 주어지게 됨
- 출력으로는 (x_train, t_train), (x_test, t_test) 형태로 주어짐
- x_train, x_test는 입력으로 들어가는 이미지가 주어지며, t_train, t_test의 경우 정답에 해당하는 label의 저장되어 있음

02. MNIST Data

- 순전파를 통해, MNIST Data를 테스트하고, 정확도를 확인

```
def get_data():
    (x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, flatten=True, one_hot_label=False)
    return x_test, t_test


def init_network():
    with open("sample_weight.pkl", 'rb') as f:
        network = pickle.load(f)
    return network


def predict(network, x):
    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']

    a1 = np.dot(x, W1) + b1
    z1 = sigmoid(a1)
    a2 = np.dot(z1, W2) + b2
    z2 = sigmoid(a2)
    a3 = np.dot(z2, W3) + b3
    y = softmax(a3)

    return y
```

WeGo

- 순전파를 통해, MNIST Data를 테스트하고, 정확도를 확인

```
x, t = get_data()
network = init_network()
accuracy_cnt = 0
for i in range(len(x)):
    y = predict(network, x[i])
    p= np.argmax(y) # 확률이 가장 높은 원소의 인덱스를 얻는다.
    if p == t[i]:
        accuracy_cnt += 1

print("Accuracy:" + str(float(accuracy_cnt) / len(x)))
```

02. MNIST Data

- Batch는 하나로 묶어서 처리할 데이터를 뜻합니다.
- 컴퓨터가 처리할 때, 1장씩 처리하는 것에 비해 효율적으로 처리가 가능
- 속도가 느린 I/O에 비해, 순수 계산의 비율을 높일 수 있음
- Batch Size가 클수록 일반화 성능이 떨어지게 되는 문제점이 있음

02. MNIST Data

- Batch는 하나로 묶어서 처리할 데이터를 뜻합니다.
- 컴퓨터가 처리할 때, 1장씩 처리하는 것에 비해 효율적으로 처리가 가능
- 속도가 느린 I/O에 비해, 순수 계산의 비율을 높일 수 있음
- Batch Size가 클수록 일반화 성능이 떨어지게 되는 문제점이 있음

```
x, t = get_data()
network = init_network()
batch_size = 100
accuracy_cnt = 0
for i in range(0, len(x), batch_size):
    x_batch = x[i:i+batch_size]
    y_batch = predict(network, x_batch)
    p = np.argmax(y_batch, axis=1)
    accuracy_cnt += np.sum(p == t[i:i+batch_size])

print("Accuracy:" + str(float(accuracy_cnt) / len(x)))
```

WeGo

02. MNIST Data

- Batch는 하나로 묶어서 처리할 데이터를 뜻합니다.
- 컴퓨터가 처리할 때, 1장씩 처리하는 것에 비해 효율적으로 처리가 가능
- 속도가 느린 I/O에 비해, 순수 계산의 비율을 높일 수 있음
- Batch Size가 클수록 일반화 성능이 떨어지게 되는 문제점이 있음

```
x, t = get_data()
network = init_network()
batch_size = 100
accuracy_cnt = 0
for i in range(0, len(x), batch_size):
    x_batch = x[i:i+batch_size]
    y_batch = predict(network, x_batch)
    p = np.argmax(y_batch, axis=1)
    accuracy_cnt += np.sum(p == t[i:i+batch_size])

print("Accuracy:" + str(float(accuracy_cnt) / len(x)))
```

WeGo

03

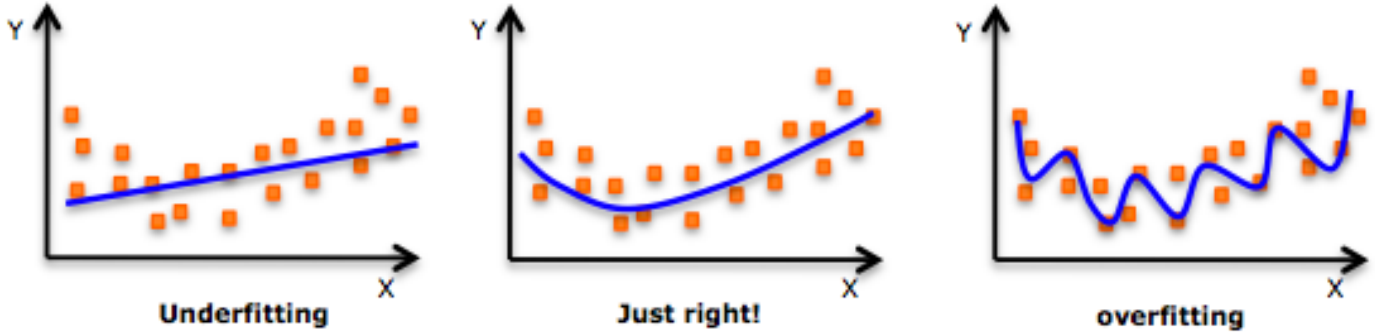
Neural Network Training

03. Neural Network Training

- Machine Learning의 핵심은 Data입니다.
- Data를 기반으로 패턴을 확인하고, 학습하는 것이 기계학습과 딥러닝입니다.
- 기존의 인식 알고리즘으로 정확한 답변을 내기가 어렵고, 특징을 명확하게 검출하기가 힘든 경우나, 사람이 말로서 쉽게 표현할 수 없는 것을 수행하는 것이 딥러닝
- 딥러닝은 종단간 기계학습(End-to-End Machine Learning)이라고 부르는 이유입니다.
- 필요한 내용은 입력과 출력, 그리고 모델입니다.

03. Neural Network Training

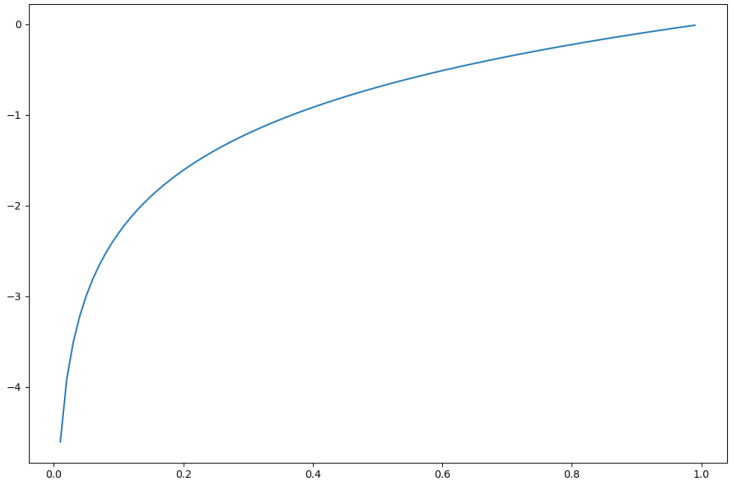
- Dataset
- 대부분의 Machine Learning은 Dataset을 Training과 Evaluation(생략가능), Test set 으로 나누게 됩니다.
- Training Set을 이용하여, 학습을 진행하며, Test Set을 이용하여, 평가를 진행합니다.
- 사람들이 원하는 것은 범용적으로 사용할 수 있는 네트워크(모델)을 원합니다.
- 이는 처음보는 Data에 대해서도 정상적으로 동작하는 것을 의미합니다.
- Training set에 대해서는 완벽하게 수행을 하지만, Test set에 대해서는 수행하지 못하는 것을 Overfitting이라고 합니다.



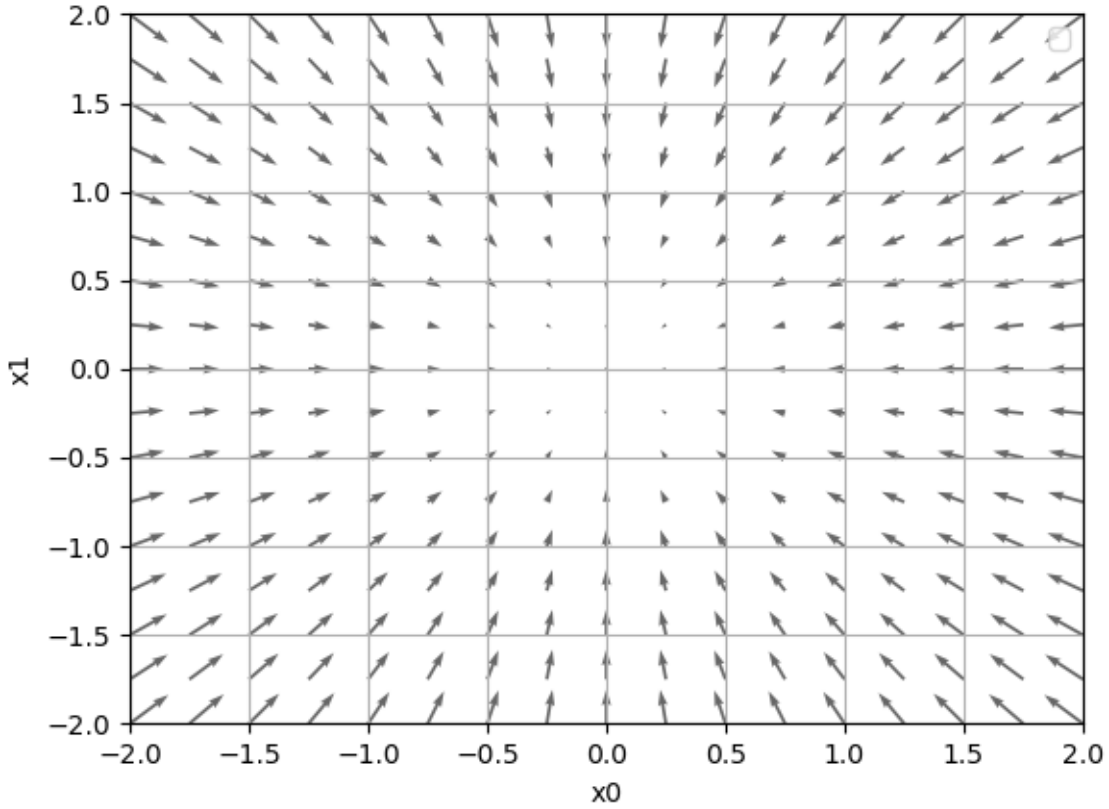
- Loss function
- 신경망의 성능의 나쁜 정도를 표현하는 지표
- 얼마나 잘 처리하지 “못”하는 지를 나타내는 지표
- 평균 제곱 오차(MSE), 교차 엔트로피(CEE)를 많이 사용
- y_k는 순전파의 결과로 나온 label, t_k는 정답에 해당하는 label

$$E = \frac{1}{2}\sum_k (y_k - t_k)^2$$

$$E = -\sum_k t_k \log y_k$$



- Gradient
- $f(x_0, x_1) = x_0^2 + x_1^2$와 같은, 다변수 함수에 대해, $(\frac{\partial y}{\partial x_0}, \frac{\partial y}{\partial x_1})$를 계산하는 것
- 각 화살표는 현재 지점에서 함수의 출력값을 가장 크게 줄이는 방향을 나타냄



```
def _numerical_gradient_no_batch(f, x):
    h = 1e-4 # 0.0001
    grad = np.zeros_like(x) # x와 형상이 같은 배열을 생성

    for idx in range(x.size):
        tmp_val = x[idx]

        # f(x+h) 계산
        x[idx] = float(tmp_val) + h
        fxh1 = f(x)

        # f(x-h) 계산
        x[idx] = tmp_val - h
        fxh2 = f(x)

        grad[idx] = (fxh1 - fxh2) / (2*h)
        x[idx] = tmp_val # 값 복원

    return grad
```

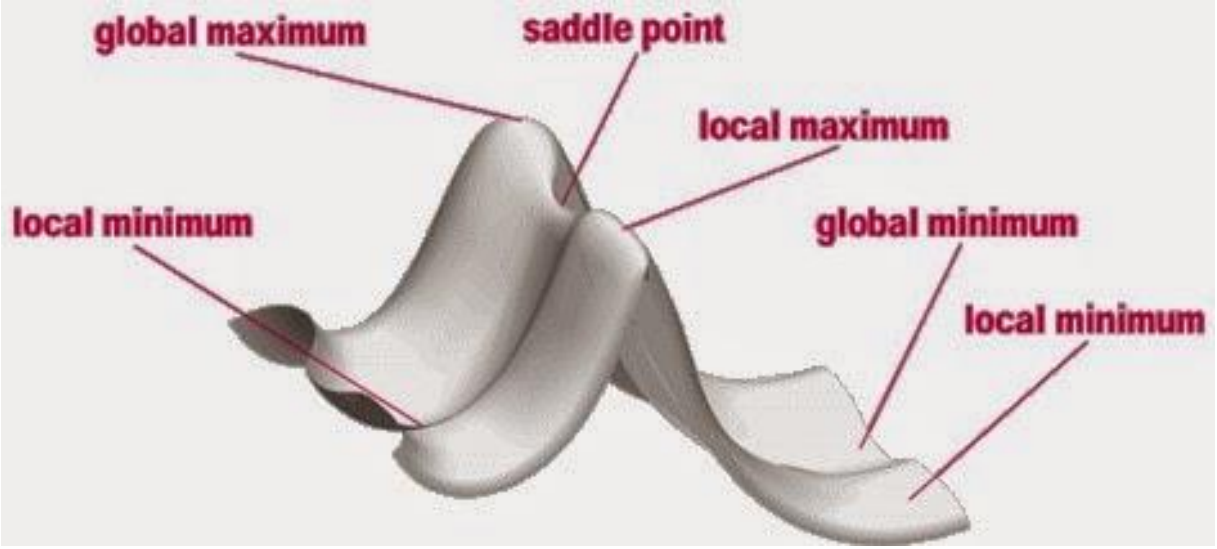
```
def numerical_gradient(f, X):
    if X.ndim == 1:
        return _numerical_gradient_no_batch(f, X)
    else:
        grad = np.zeros_like(X)

        for idx, x in enumerate(X):
            grad[idx] = _numerical_gradient_no_batch(f, x)

        return grad
```

```
def function_2(x):
    if x.ndim == 1:
        return np.sum(x**2)
    else:
        return np.sum(x**2, axis=1)


def tangent_line(f, x):
    d = numerical_gradient(f, x)
    print(d)
    y = f(x) - d*x
    return lambda t: d*t + y
```

```
if __name__ == '__main__':
    x0 = np.arange(-2, 2.5, 0.25)
    x1 = np.arange(-2, 2.5, 0.25)
    X, Y = np.meshgrid(x0, x1)

    X = X.flatten()
    Y = Y.flatten()

    grad = numerical_gradient(function_2, np.array([X, Y]) )

    plt.figure()
    plt.quiver(X, Y, -grad[0], -
grad[1],  angles="xy",color="#666666")#,headwidth=10,scale=40
,color="#444444")
    plt.xlim([-2, 2])
    plt.ylim([-2, 2])
    plt.xlabel('x0')
    plt.ylabel('x1')
    plt.grid()
    plt.legend()
    plt.draw()
    plt.show()
```

- 경사 하강법(Gradient Descent Method)
- 앞서 계산한 Gradient를 이용하여, 함수의 최소값을 찾아내는 방법
- Gradient를 이용하므로, 기울기가 0인 장소를 찾아내는 것이 목표

→ 이 때문에, Global minimum 값이 아닌, local minimum에 갇히거나, saddle point에서 정지할 문제도 있음



- 경사 하강법(Gradient Descent Method)
- $x_0 = x_0 - \eta \frac{\partial f}{\partial x_0}, x_1 = x_1 - \eta \frac{\partial f}{\partial x_1}$
- η는 Learning Rate를 의미하며, 너무 작거나 클 경우, 좋은 장소에 도달하기가 힘듦
- 일반적으로 Learning Rate을 조절하며, 학습이 제대로 되고 있는지 확인

- 경사 하강법(Gradient Descent Method)
- $x_0 = x_0 - \eta \frac{\partial f}{\partial x_0}, x_1 = x_1 - \eta \frac{\partial f}{\partial x_1}$
- η는 Learning Rate를 의미하며, 너무 작거나 클 경우, 좋은 장소에 도달하기가 힘듦
- 일반적으로 Learning Rate을 조절하며, 학습이 제대로 되고 있는지 확인

```
def gradient_descent(f, init_x, lr=0.01, step_num = 100:
    x = init_x

    for i in range(step_num):
        grad = numerical_gradient(f, x)
        x -= lr * grad
    return x
```

- 경사 하강법(Gradient Descent Method)을 통해 최소값 계산하기
- $f(x_0, x_1) = x_0^2 + x_1^2$의 최소값을 경사하강법을 이용하여 계산(초기값 -3, 4)

- 경사 하강법(Gradient Descent Method)을 통해 최소값 계산하기
- $f(x_0, x_1) = x_0^2 + x_1^2$의 최소값을 경사하강법을 이용하여 계산(초기값 -3, 4)

```
def function_2(x):
    return x[0]**x + x[1]**2

init_x = np.array([-3.0, 4.0])
gradient_descent(function_2, init_x=init_x, lr=0.1, step_num=100)
```

lr을 조정하여, Learning Rate의 중요성을 확인할 수 있음
lr = 10.0일 경우, 발산
lr = 1e-10일 경우, 최소값에 도달할 수 없음

WeGo